OIPE JCIS5
OCT 1 5 2004
PATENT & TRADEMARK OFFICE

T FW

Attorney's Docket No.: 13913-087001 / 2002P10032 US02

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

RECEIVED
NOV 1 2 2004
Technology Center 2100

Applicant	:	Markus Cherdron et al.	Art Unit	:	2123
Serial No.	:	10/676,837	Examiner	:	Unknown
Filed	:	September 30, 2003			
Title	:	USING INTERRELATED DATA AT RUN TIME			

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL OF PRIORITY DOCUMENT UNDER 35 USC §119

Applicant hereby confirms his claim of priority under 35 USC §119 from the following application(s):

·European Patent Office Application No. 02024242.6 filed October 31, 2002

A certified copy of each application from which priority is claimed is submitted herewith.

Please apply any charges or credits to Deposit Account No. 06-1050.

Respectfully submitted,

Date: 10-11-04

[illegible]
Elissa Y. Wang
Reg. No. 48,668

Fish & Richardson P.C.
500 Arguello Street, Suite 500
Redwood City, California 94063
Telephone: (650) 839-5070
Facsimile: (650) 839-5071

RECEIVED
NOV -3 2004
OIPE/JCWS

50240992.doc

CERTIFICATE OF MAILING BY FIRST CLASS MAIL

I hereby certify under 37 CFR §1.8(a) that this correspondence is being deposited with the United States Postal Service as first class mail with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

October 12, 2004

Emma Durell
Signature

Emma Durrell
Typed or Printed Name of Person Signing Certificate

10-15-2004
U.S. Patent & TMOfc/TM Mail Rcpt Dt. #72

Europäisches Patentamt | European Patent Office | Office européen des brevets

Bescheinigung | Certificate | Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. | Patent application No. | Demande de brevet n°

02024242.6

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

[illegible]

R C van Dijk

Anmeldung Nr:
Application no.: 02024242.6
Demande no:

Anmeldetag:
Date of filing: 31.10.02
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

SAP Aktiengesellschaft
Neurottstrasse 16
69190 Walldorf
ALLEMAGNE

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.
If no title is shown please refer to the description.
Si aucun titre n'est indiqué se referer à la description.)

Computer system and methods for using interrelated data at runtime

In Anspruch genommene Prioriät(en) / Priority(ies) claimed /Priorité(s) revendiquée(s)
Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

US/30.09.02/US 414985 P

Internationale Patentklassifikation/International Patent Classification/
Classification internationale des brevets:

G06F9/44

Am Anmeldetag benannte Vertragstaaten/Contracting states designated at date of filing/Etats contractants désignées lors du dépôt:

AT BE BG CH CY CZ DE DK EE ES FI FR GB GR IE IT LI LU MC NL PT SE SK TR

COMPUTER SYSTEM AND METHODS FOR USING INTERRELATED DATA AT RUNTIME

Field of the Invention

The present invention relates to electronic data processing in general, and particularly to application programming.

Background of the Invention

In the generally accepted model view controller (MVC) design pattern used for developing application programs, the model represents the core of such an application program. The model can have multiple views, where each view displays information about the model to a user. A controller of the model receives events, for example, raised by a user interacting with a view to manipulate the model. The model can have multiple controllers and a controller can relate to multiple views. The model and the controller typically include application code. When changes occur in the model, the model updates all of its views. So called data binding is used for data transport between the view and its model or controller. For example, a table view can be defined to display data of a corresponding table that is stored in the model or controller. The table is used as data source for the table view (data binding). For example, the table view can be replaced by a further view, such as a linked list, that binds against the same table. In this case the further view displays the table data without changing anything in the controller or the model.

When building a software application, typically some predefined relationships exist between various data used by the application. For example, the relationships are defined through dependencies in a relational database. However, for some data predefined relationships do not exist, for example, when no relationship is defined in a database or when it is data that refer to the model on the one hand and to the view on the other hand. Therefore, usually a major part of application coding is used to define the corresponding relationships and to enable the data transport, for example, from the model to the view.

Further, at a given point in time an application has a specific state that reflects the current status of the interaction of the user with the application (e.g., on which view is the cursor of the application and which row of a specific table in the view has been selected by the user). Typically, an application developer has to write application coding to memorize and administrate the state (e.g., by using state variables).

Further, when the user of a client-server system interacts with the client, typically the client sends a request to the server to rebuild a current page and the server sends the rebuilt page to the client. This may cause an unpleasant effect for the user in the form of a flickering picture on a display device of the client. Some client-server systems support mechanisms to rebuild only mandatory components of the page and send only the corresponding delta information to the client to reduce flickering. However, to determine the delta information application specific coding has to be developed on both sides, the client and the server.

Summary of the Invention

To alleviate the problems of prior art systems the present invention provides methods, systems and computer program products to provide an extended MVC design pattern for structuring data of an application in contexts that correspond to controllers.

A computer system that is used to develop and/or run applications according to the extended MVC pattern includes a model, at least one view and at least one controller. The model can have multiple controllers. A controller can control a view or further controllers. The at least one view presents the model. The at least one controller relates to the at least one view and manipulates the model. The computer system further includes at least one context. The at least one context relates to the at least one controller and declares data of the application. The at least one context further declares data hierarchies that describe relationships between the data.

Different types of context can be defined. A view context is assigned to a view controller. The lifetime of the view context/controller equals the lifetime of the corresponding view. In other words, the view context lives as long as the corresponding view is displayed. A custom context is assigned to a custom controller. A custom controller is one that has no views. The lifetime of a custom context can span the lifetime of multiple views. Therefore, a customer context can store data (e.g., data defining the interaction status) that are needed in multiple views.

Context hierarchies can be defined. A view context can reference data from a custom context. This allows an efficient componentisation of the application because data can be accessed from a view through corresponding references to custom contexts instead of copying the data each time to the corresponding view context.

It is an effect of the present invention that less memory is needed to store an interaction status because redundant data storage is eliminated.

It is a further effect of the present invention that data are stored only once in an application. Therefore, program code redundancy that originates from using multiple variables for the same data in prior art systems is eliminated. This improves the data consistency within an application.

It is a further effect of the present invention that the data relationships are defined in a declarative way. Therefore, specific functions can be implemented without application specific program code. Examples of these functions are:

filtering by using the data relationships; or

identifying which data have been changed to determine the delta that needs to be sent from a server to a client.

The aspects of the invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims. It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory only and are not restrictive of the invention as described.

Brief Description of the Drawings

FIG. 1	is a simplified block diagram of a computer system that implements an embodiment of the extended MVC design pattern;
FIG. 2	illustrates an example of a structure of a context at design time and at runtime;
FIG. 3	illustrates the context at runtime as a set of data instances;

FIG. 4	illustrates an example of a node selection within the context at runtime;
FIGS. 5A,5B	illustrate two alternative runtime implementations of context data instances;
FIG. 6	illustrates an example of context lifetimes for various context types;
FIG. 7	illustrates mapping of contexts according to the present invention; and
FIG. 8	illustrates a specific example of mapping contexts.

Detailed Description of the Invention

The present invention introduces the concept of context to the MVC design pattern. This will be referred to as extended MVC design pattern.

FIG. 1 is a simplified block diagram of a computer system 900 that implements an embodiment of the extended MVC design pattern.

The extended MVC design pattern provides a context as a structured storage place for data that relates to a controller. A context instance 304 relates (dashed line) to a controller instance 302. Context instances and controller instances will be referred to as contexts and controllers, respectively. The controller 302 can manipulate a model 301 in response to an interaction of a user 10 with the computer system 900. There can be further controllers (e.g., further controllers 302-a, 302-b, 302-c) for manipulating the same model 301. The further controllers can have further contexts 304-a, 304-b, 304-c that relate (dashed lines) to the further controllers, respectively. The model 301 can have multiple views (e.g., views 303, 303-a, 303-b) that present the model to the user 10. When the model 301 gets modified by at least one of its controllers it updates all of its views. Each view relates (dashed lines) to a controller. There can be controllers (e.g., controller 302-c) that do not relate to any view. In one embodiment, a controller can relate to multiple views.

FIG. 2 illustrates an example of a structure of a context 304 at design time and at runtime. In general, structure elements of the design time context structure are different from structure elements of the runtime context structure.

An example of a design time context structure is a node hierarchy, wherein the structure elements of the node hierarchy can be nodes and attributes. The root-node of the node hierarchy represents the context itself. For example, the child nodes of the root node can be defined by the application. Child nodes of the root node will also be referred to as independent nodes. Child nodes of independent nodes depend on their corresponding parent node and will also be referred to as dependent nodes.

A node has a node type. Examples of node types are value nodes and model nodes. A value node can maintain, that is, store and administrate, its own application data (transient application data). The data can be, for example, scalar data, tables or structures. A model node includes a reference to a corresponding model that persists application data.

The parent node can also have attributes. Each child node can include an arbitrary tree structure that includes further child nodes and/or attributes. Attributes are leaves in the tree structure. Attributes represent, for example, scalar data types, such as strings and integers or Java types (e.g., java.util.Date).

In the example of FIG. 2, at design time, the context 304 includes the independent node PN that includes the two attributes A1, A2 and that is the parent node of the dependent nodes CN1, CN2. The second dependent node CN2 has two further attributes A3, A4. This structure defines a first node element 701 for the parent node PN and a second node element 702 for the second child node CN2. The first node element 701 includes information about the context structure with regards to the parent node PN. In other words, it summarizes all information that is available at the context structure level that is under the level of the parent node PN. The second node element 702 includes information about the context structure with regards to the second dependent node CN2. The context structure implies that the second node element 702 depends on the first node element 701.

At runtime, structure elements (e.g., nodes) represent a set of data instances. Nodes provide type information about object instances that are maintained by the node. Each node can have a node collection, wherein each element of the node collection has the same node element type.

In the example of FIG. 2, at runtime, the parent node PN has a first node collection 401 that includes multiple runtime instances of the first node element 701. Each runtime instance of the first node element 701 can have a second node collection 402 of multiple runtime instances of the second node element 702. A node collection can be empty or has at least one instance of the corresponding node element.

A node collection has a cardinality and a node collection type, such as list, tree, set or collection.

The node collection cardinality (cf. table 2) and the node collection type (cf. table 1) can be declared at design time. An evaluation mechanism can be used to automatically evaluate the node collection of a child node at runtime when its parent node changes.

Table 1: Examples of node collection types

Value	Meaning
Collection	forward-only iterator (cursor) without absolute positioning
Set	no duplicates, forward-only iterator without absolute positioning
List	duplicates allowed, position available, list iterator, absolute positioning (indexed access)

The application can use the cardinality of a node collection to restrict possible operations on a node (e.g., prohibit indexed access to a node that has at most one node collection element).

Table 2: Examples of the cardinality of a node collection

Value	Meaning
0..1	node collection can be empty, contains at most one element
1..1	node collection always contains exactly one element.
0..n	node collection can be empty or contain any number of elements
1..n	node collection always contains at least one element.

The content of a node collection can be determined in various ways.

The node values of independent nodes can be set by initializers or event handlers or can be set through a supply function. The supply function is called when the node is accessed. To access a node, for example, the node is queried for its data by application code or by a user interface (UI) element (of the view) that is bound to the node.

Dependent nodes can get their values by using a supply function. For example, the node collection of a dependent node can become obsolete when a selection of its parent node changes. In this case the dependent node is recalculated, that is, the content of its node collection is determined on a subsequent access. In another example a representation instance is created for each dependent node of a parent node. The values of the representation instances are calculated when the corresponding parent node is accessed. In other words, using representation instances enables a "load data on demand" or a "unload data when not needed" mechanism. Therefore, memory is used in an efficient manner.

The content of a node collection can also be explicitly set to a state, such as "invalid" or "unfilled". When the node is accessed the next time, the node collection content is determined again. This can be used to force a re-read of modified data when the modification (e.g., in the model) was not visible to the application runtime.

FIG. 3 illustrates the context 304 at runtime as a set of data instances. The nodes of the context at runtime represent a framework-managed set of data instances (e.g., a java.sql.RecordSet). For example, data instances are returned 50 from a database or backend system 901 in response to a query (e.g., a structured query language (SQL) query) that is sent 40 from the computer system 900 to the database/backend system 901 when a node is accessed, for example, by an application. Examples of backend systems are Enterprises Resource Planning systems, Customer Relationship Management systems, web server systems providing web services or any other system that stores application data. Accessing a node means requesting data from the corresponding model. This can result in a corresponding query request from the model to the database/backend system 901. Nodes provide type information about object instances that are maintained by the node. The type information can also be derived from the model. For example, if the parent node PN corresponds to a customer, its node collection 401 can include all orders for this customer. When the application accesses the parent node PN the computer system 900 can sent 40 a query to retrieve all orders of the customer from the corresponding database/backend system 901, such as a sales and distribution (SD) system or a customer relationship management (CRM) system. The retrieved orders (data instances) are then returned 50 to the computer system 900 context 404 to fill the corresponding data of elements of the node collection 401.

FIG. 4 illustrates an example of a node selection 501 within the context 304 at runtime.

A node PN can maintain a node selection 501 within a node collection 401. Node selections are illustrated in FIG. 4 by a grid pattern for each element of the node collection that belongs to the node selection. The node selection 501 is a designated subset (one or more elements) of the node collection 401 of the node PN. The node selection 501 has a cardinality that is controlled by the cardinality of the selected nodes declared at design time (cf. table 3, below). One specific element that plays a special role amongst the elements of the node selection will be referred to as the lead selection element.

For example, if the node PN corresponds to a specific customer, the first node collection 401 can include all orders of the customer.

The lead selection of the node collection can be by default the first order of the customer. In this case, the second node collection 402 can include all order items of the selected order.

Table 3: Examples of the cardinality of a node selection in dependence of the node (collection) cardinalities of the corresponding node elements of the selection

Value	Meaning	Node Cardinality
0..1	single selection (≅ lead selection), can be empty	Any
1..1	single selection (≅ lead selection), always contains one element	only 1..1, 1..n
0..n	multiple selection; can be empty; if not empty one element is designated as the "lead selection"	only 0..n, 1..n
1..n	multiple selection. One selected element is designated as the "lead selection"	only 1..n

If the node selection is not empty at runtime, one of the elements of the node selection is designated as the lead selection element. The lead selection element can be accessed from controller code. UI elements can be bound against the attributes of the lead selection element and the content of a child node depends on the lead selection element of its parent node. For example, the node selection 501 can correspond to a selection that results from an user action (e.g., the user selects the second order out of a list of orders.) This automatically triggers an update of the second node collection 402 with, for example, all order items of the second order. The second node collection 402 can have a further node selection 502. A node selection can also include multiple elements of the corresponding node collection.

Node selection and lead selection element are bindable node properties in the sense that UI elements can represent a node selection (e.g., as selected lines in a table control) and also modify it (e.g., selecting/deselecting an item in a table control adds/removes the corresponding element to/from the node selection). Node selections can exist on their own. A selection made by a user can be represented as a node selection and a node selection can be visualized in a UI element.

A context can include a flat set of child nodes (independent nodes) each one independent from the others. Each independent node can have further child nodes (dependent nodes). While the content of independent nodes can be defined by the application, the content of a dependent node depends on the lead selection element of its parent node. The application defines how the content of the dependent node depends on the parent node's lead selection element by specifying a corresponding supply function. For example, a supply function can be used in case a specific order (e.g., node selection 501) of a customer is selected and only order items that are not on stock should be included in the second node collection 402. In other words, the relationships between data that are declared in the context 304 at design time can be used to filter data at runtime.

For example, the supply function can be defined in such a way that it always returns the same value for the same selected node element and does not take into account changes in the returned data. In other words, the application runtime can decide not to call a supply function again with the same arguments when it is called a second time within the lifetime of the application.

For example, when a parent node (e.g., an order) is bound to a new node collection, the content of all of its child nodes (e.g., order items) becomes "invalid". When a node is accessed and its content (node collection) is "invalid", its content is determined again, for example, by calling a corresponding supply function 601 to supply content for the node.

Supply functions can be declared as methods in the corresponding controller 302. The following pseudo code shows an example of the signature of a supply function:

Pseudo code example:

```
Collection supplyFunction(Node node, NodeElement
parentElement);
```

When the application is generated, program code is generated that calls the declared method when content for a node is to be supplied 60.

Embodiments of a supply function can have one or more of the following features:

- Node elements included in a returned node collection match the type of the corresponding node (e.g., a node element created from the node or from a mapped node or from a corresponding model class, if the node is a model node)

- The supply function returns enough data to match the declared cardinality of the node.
- The returned node collection depends on parameters of the supply function. The supply function is called a second time within the lifetime of an application when at least one of the parameters is changed.
- The supply function can also be loaded on demand by the application.

FIGS. 5A and 5B illustrate two alternative runtime implementations of context node data instances of a context 304-a.

In a first implementation (cf. FIG. 5A), a dependent node (e.g., node B) can be represented as a single node instance whose node collection changes whenever the parent node's (e.g., node A) node collection or lead selection element changes. For example, for a single node instance, content (node collection) can be maintained for the current lead selection of the parent node only. This reduces the amount of used system resources, such as main memory, and it enables static binding. Static binding means that the node binds to a "class" of the node instead of binding to a named node instance. A node according to the first implementation will be referred to as a singleton node.

FIG. 5A shows an example of a context structure of context 304-a at design time. Node A has a node element NE(A), node B has a node element NE(B) and node C has a node element NE(C), wherein each element includes child nodes and/or attributes. At runtime, in case of a singleton node implementation, a node collection NC(B) of node element NE(B) instances is only maintained for the lead selection of the node collection NC(A).

Further, a node collection NC(C) of node element NE(C) instances is only maintained for the lead selection of the node collection NC(B).

In a second implementation (cf. FIG. 5B) a single node instance of the node (e.g., node B) exists for each instance in the parent node collection (e.g., node collection NC(A)). All single node instances can be accessed directly. For example, a runtime implementation can create and fill single node instances lazily (by loading data on demand) to reduce resource usage. In the second implementation an application can also access data of child nodes that do not correspond to the parent node's lead selection element (e.g., read address fields for business partner No. 5 instead of the address fields for the currently selected business partner No. 3). A dependent node according to the second implementation will be referred to as a non-singleton node.

FIG. 5B is based on the context structure 304-a at design time as described under FIG. 5A. It shows an example of a runtime structure of context 304-a according to the second implementation. Each instance in node collection NC(A) can have a node collection NC1(B) to NC3(B). Further, each instance of node collections NC1(B) to NC3(B) can have a node collection NC1(C) to NC5(C). Empty node collections are not shown in the example.

The information if a node is a singleton or non-singleton node can be stored in a node property "singleton" (cf. table 4, below).

If a non-singleton node acts as the parent node of a singleton node, the singleton node is not a singleton node with regards to the context. That is, for each instance of the non-singleton parent node there exists one instance of the singleton child node. If the child node is a singleton node with regards to the context, then its parent node may change depending on its grandparent node's lead selection element.

A framework keeps references to all created instances of a child node until the parent node's collection changes. This enables a client in a client-server computer system to remember data from previously received child node instances and modify this data later. The server keeps this data and has, at all times, a consistent picture of which data is in the current context (= context of the current view).

Table 4: node property singleton

Value	Meaning
True	a single instance of the node exists per parent node and the content of the node changes when the parent node's lead selection element changes.
False	one instance of the node exists per node element in the parent node's node collection. The content of an instance does not change.

All instances of a child node can be accessed through a typed context application programming interface (API).

If the parent node is a singleton node, only a single instance exists and can be accessed and its content depends on the parent node's node collection and lead selection element.

For example, at design time, a tree structure is declared including an independent node "Customers" that has a child node "Orders" and the child node "Orders" has a further child node "OrderItems". Each customer can have multiple orders and each order can have multiple items. This is reflected in a corresponding context by declaring child nodes belonging to each element of the parent node so that each element has a collection of its own.

FIG. 6 illustrates an example of context lifetimes for various context types.

There are at least two types of controllers and correspondingly two types of contexts: view controllers/view contexts and custom controllers/custom contexts.

A view controller relates to a corresponding view. The lifetime of the view controller equals the lifetime of the corresponding view, that is, the time the view is displayed. A view context relates to the view controller and has the same lifetime. UI elements of the view can bind to the view context. When executing an application (e.g., APPLICATION A) that is built according to the extended MVC design pattern, typically a sequence of multiple views (e.g., VIEW 1, VIEW 2, VIEW 3, VIEW 4) is presented to a user. The user interacts with the application program through the various views. The various views can raise events that cause the related view controllers to determine which view is presented when and where. That is, some views and, therefore, the related view contexts can have a short lifetime.

In the example of FIG. 6, APPLICATION A starts at TA1 and ends at TA2. When the application starts, VIEW 1 and VIEW 2 are presented to the user simultaneously. At TV1, the corresponding view controllers determine that the presentation of VIEW 1 and VIEW 2 needs to be replaced by a presentation of VIEW 3. At TV2, the corresponding view controllers determine that the presentation of VIEW 3 needs to be replaced by a presentation of VIEW 4. The views VIEW 1 to VIEW 4 relate to the view contexts VIEW CONTEXT 1 to VIEW

CONTEXT 4. That is, the data that is stored in each view context has the same lifetime as the view that binds to the data.

Some data need to be stored over the lifetime of multiple views. For this purpose, a custom context can be defined. A custom context relates to a custom controller of the model. For example, a custom controller is implemented as view independent, application process oriented coding. The lifetime of a custom context can be defined in such a way that it spans the lifetime of multiple views.

In the example of FIG. 6, CUSTOM CONTEXT I is defined to span the lifetime of the views VIEW 1 to VIEW 3. CUSTOM CONTEXT II is defined to span the lifetime of the views VIEW 3 and VIEW 4.

A specific example of a custom context is an application context, which lives over the lifetime of the application, that is, over the sequence of all views of the application. However, in the case of a custom context, the application decides about its lifetime, whereas in the case of an application context, a framework can decide about the lifetime of the application context because the framework knows when an application starts (TA1) and when it ends (TA2). Therefore, the framework can control an application controller that is assigned to the application context.

FIG. 7 illustrates mapping of contexts according to the present invention.

Because UI elements (e.g., UI elements 951, 952) of views (e.g., VIEW 1, VIEW 2) that are used in a user interface (UI) 950 bind 81, 82 to view contexts (e.g., VIEW CONTEXT 1, VIEW CONTEXT 2) and long living data can reside in custom contexts (e.g., CUSTOM CONTEXT I), an embodiment of the present invention enables mapping 91, 92 of nodes/attributes of view contexts or custom contexts to nodes/attributes of custom contexts. In other words, nodes and attributes of view contexts or custom contexts can reference type-compatible nodes and attributes in other custom contexts. Nodes can also be mapped to other nodes within the same context. Node mapping reduces the need for copying data between several contexts by enabling a node N1 of a first context (e.g., a view context, such as VIEW CONTEXT 2, or a custom context) to reference 91 a node N1' of a second context (e.g., a custom context, such as CUSTOM CONTEXT I, or an application context), where the node N1' of the second context has or references the data. The same is true for attributes.

Therefore, the data can be manipulated in a custom/application context and each view context that references the custom/application context provides its view with the current data stored in the custom/application context. Mapping contexts can span multiple context levels. That is, a custom context can reference a further custom context. Therefore, context hierarchies can be created (cf. FIG. 9).

For example, related data can be collected in a dedicated custom context. The binding to this data is implemented by using a view context that is mapped to the custom context accordingly.

The extended MVC pattern enables an application developer to quickly modify an application while maintaining consistency of the application. For example, in some cases rearrangement of views or UI elements can be achieved without modifying the corresponding controller code. This provides a way for an application developer to better structure applications in light of potential functional enhancements or changes. For example, reusing a field that already exists on one view in other views can be achieved by defining the corresponding mapping while the corresponding controller code stays valid.

The following examples explain various features of context mapping that can be implemented with the present invention.

First example:

If a node M ("Mapped Node") is mapped to a node O ("Origin Node"), node M maps its node collection to node O's node collection. The node selections of nodes M and O can be mapped. Node M can also maintain its own node selection on node O's node collection.

For example, the node collection cardinality of node M equals that of node O (e.g., by inheritance).

The selection cardinality can be inherited from origin node O. Node M can override the node cardinality inherited from node O.

If node O is a singleton node, node M is a singleton node, too. If node O is a non-singleton node, node M can be a singleton or non-singleton node. If node M is a non-singleton node it shares the same parent node collection with node O. If node M is a singleton node, then the collection of node M follows the instance of node O that belongs to the lead selection of node O's parent node.

For mapped nodes, the content of the node collection can be defined by the node collection of the origin node.

Second example:

An independent node can always be mapped. It can be mapped to any other node in the same context or to any other node in another custom context (as long as no cycle is formed with regards to parent-child and mapping relationships).

A child node of a mapped node can be unmapped. In this case its content can be determined by the supply function mechanism.

When a parent node is mapped to a further parent node, a child node of the parent node can be mapped to a further child node of the further parent node. In other words, if node W is a child of node X and node Y is a child of node Z, node W can be mapped to node Y if node X is mapped to node Z.

If a child node of a mapped node is mapped to a further child node of the corresponding origin node, then either the mapped node maps to the node selection of the origin node or the origin node is a non-singleton node. This avoids a conflict between the dependencies implied by the parent/child relationship and the mapping relationship that results from mapping a selection of a child node of an unmapped node.

FIG. 8 illustrates a third, specific example of mapping contexts.

Two windows 950-1, 950-2 can be displayed at runtime on a client of a client-server system. For example, the windows are part of a user interface of an application and are displayed on a conventional display device (e.g., monitor) of the client. A page that is displayed may include one or more view assemblies.

The first window 950-1 displays view assembly MAIN that includes view A and view B. The second window displays view assembly POP UP that includes view D. The following description refers to definitions and declarations at design time. The views in the view assemblies include UI elements which are bound to the view contexts of the corresponding views. The binding is illustrated by bended arrows with a bullet point at the origin of the arrows. UI elements of views A, B, D are bound to view contexts A, B, D, respectively. The UI element in view A is a table having two columns. The four UI elements of view B can be display/input fields that have a relationship to the table of view A. The UI elements of view D correspond to a title of the pop up and four further input/display fields.

The view contexts A, B, D include node/attribute hierarchies for maintaining the data of the corresponding view. Nodes and attributes can derive their state from nodes/attributes of custom contexts (e.g., custom contexts 1, 2) that belong to controllers (e.g., custom or application controllers) other than the corresponding view controllers. This enables maintenance of the data without redundancies. Further, it can be used for a method for synchronizing data dependencies amongst multiple views.

In the example of FIG. 8, view context A and view context B include one independent node each, which is illustrated as the top-level node of the corresponding context structure. The independent node of view A holds information about which record set is to be used for the table and about the current position within the record set. Both independent nodes are mapped to the corresponding independent node in custom context 1. This means that view A and view B share a common data source (e.g., the record set) provided by the commonly used node of custom context 1. Therefore, the record set displayed in the table of view A is also used as the underlying record set for view B. View A displays two columns of the record set, whereas view B displays three fields of a selected row of the record set. This is represented by the UI elements illustrated by a grid pattern. The three fields in view B can also serve as input fields to update the underlying record set. View B displays a further field not related to the record set.

The declaration of data relationships through contexts leads to redundancy free data transport from the server to the client and allows the application to synchronize the table of view A with the input in view B. It also allows an application developer to use the current selection in a custom controller without needing to know how the selection was made (e.g., by using a table view UI element, or a dropdown list UI element or any other UI Element able to make a selection in a list). This decreases the dependency of application logic from presentation logic.

Context mapping can also be used to open a menu/list (e.g., view D in the view assembly POP UP), which can display data based on the current selection. No transport code is necessary and no selection parameters need to be passed.

In the example of FIG. 9, the last two attributes of view context D are mapped to corresponding attributes of custom context 2. Because the last attribute of view context B maps to the same attribute of custom context 2 as the next to last attribute of view context D, the content of the upper input/display field in view B equals the content of the upper input/display field in view D. No transport code for transporting data from view B to view D is necessary to achieve this.

The last attribute of view context D is mapped to the last attribute of custom context 2 which again is mapped to the next to last attribute of custom context 1. This illustrates that multilevel context hierarchies can be built. Multilevel context hierarchies are useful to package data according to their lifetime because, as explained in reference to FIG. 6, each context can have a different lifetime. Storing data only once in the context hierarchy and using mapping to access the data through multiple levels of the context hierarchy avoids redundant storage of data and, therefore, reduces main memory consumption.

An embodiment of the present invention can be implemented by using a computer system that has at least a memory and a processor. The computer system can communicate with further computer systems over a network (e.g., a wide area network (WAN), a local area network (LAN), the Internet.) A computer program product that can be loaded into the memory of the computer system includes instructions that when executed by the processor causes the computer system to perform steps according to the present invention.

The invention can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. The invention can be implemented as a computer program product, i.e., a computer program tangibly embodied in an information carrier, e.g., in a machine-readable storage device or in a propagated signal, for execution by, or to control the operation of, data processing apparatus, e.g., a programmable processor, a computer, or multiple computers. A computer program can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program can be deployed to be executed on one computer or on multiple computers at one site or distributed across multiple sites and interconnected by a communication network.

Method steps of the invention can be performed by one or more programmable processors executing a computer program to perform functions of the invention by operating on input data and generating output. Method steps can also be performed by, and apparatus of the invention can be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application-specific integrated circuit).

Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read-only memory or a random access memory or both. The essential elements of a computer are a processor for executing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto-optical disks, or optical disks. Information carriers suitable for embodying computer program instructions and data include all forms of non-volatile memory, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in special purpose logic circuitry.

To provide for interaction with a user, the invention can be implemented on a computer having a display device, e.g., a CRT (cathode ray tube) or LCD (liquid crystal display) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input.

The invention can be implemented in a computing system that includes a back-end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front-end component, e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation of the invention, or any combination of such back-end, middleware, or front-end components. The components of the system can be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network ("LAN") and a wide area network ("WAN"), e.g., the Internet.

The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.

The invention has been described in terms of particular embodiments. Other embodiments are within the scope of the following claims. For example, the steps of the invention can be performed in a different order and still achieve desirable results.

Claims

1. A computer system (900) comprising a model (301), at least one view (303) and at least one controller (302); the at least one view (303) presenting the model (301); the at least one controller (302) relating to the at least one view (303) and manipulating the model (301);
characterized in that:
the computer system (900) further comprises at least one context (304) that relates to the at least one controller (302);
the at least one context (304) provides data of an application to the view (303), at runtime; and
the at least one context (304) comprises at least a first node collection (401) that comprises at least one runtime instance of a node (PN) of the context (304).

2. The computer system (900) of claim 1, wherein the at least one runtime instance of the node (PN) has a first node element type that is predefined by a first node element (701) of the node (PN) at design time.

3. The computer system (900) of claim 2, wherein the first node element (701) comprises a child node (CN2) of the node (PN) and the child node (CN2) has a second node element (702) that is pre-configured at design time.

4. The computer system (900) of claim 3, wherein the at least one runtime instance of the node (PN) has at least a second node collection (402) that comprises at least one runtime instance of the child node (CN2) having a second node element type that corresponds to the second node element (702).

5. The computer system (900) of any one of the claims 1 to 4, wherein the first node collection (401) comprises at least one further runtime instance of the node (PN) that has the first node element type and the further runtime instance of the node (PN) has no further node collection.

6. The computer system (900) of any one of the claims 1 to 4, wherein the first node collection (401) comprises at least one further runtime instance of the node (PN) that has the first node element type and the further runtime instance of the node (PN) has at least a further node collection that comprises at least one runtime instance of the child node (CN2) having the second node element type.

7. The computer system (900) of claim 5 or 6, wherein a node selection (701) comprises the at least one runtime instance of the first node collection (401).

8. The computer system (900) of claim 7, wherein the node selection (701) is changed to comprise the further runtime instance of the first node collection (401) and an evaluation mechanism evaluates the second node collection (402) resulting in a state of the second node collection (402).

9. The computer system (900) of claim 8, wherein the state is selected from the group of valid, invalid and unfilled.

10. The computer system (900) of claim 9, wherein a supply function (601) supplies (60) content to the second node collection (402), if the state of the second node collection (402) is invalid or unfilled.

11. The computer system (900) of claim 10, wherein the supply function (601) is implemented as a method of the controller (302).

12. The computer system (900) of claim 10 or 11, wherein the supply function (601) supplies at least one new runtime instance that has the second node element type.

13. A method for supplying data to a view (303) presenting a model (301); the view (303) having at least one user interface (UI) element and relating to a controller (302) for manipulating the model (301); the method comprising the following steps:
using a context (304) that relates to the controller (302);
creating a runtime context structure of the context (304) by using a design time context structure of the context (304) being the basis for runtime behaviour, wherein the design time context structure comprises a structure element (PN) to which the UI element binds; and
providing a supply function (601) to supply (60) at least one runtime instance of the structure element (PN) to the view (303) at runtime.

14. The method of claim 13, wherein the supply function (601) is implemented in the controller (302) that relates to the view (303).

15. The method of claims 13 or 14, wherein the creating step creates a node collection (401) of the structure element (PN) at runtime and the providing step adds the at least one runtime instance to the node collection (401).

16. The method of claim 15, wherein each runtime instance of the node collection (401) has a node element type that corresponds to a node element (701) of the structure element (PN).

17. A method for accessing application data by an application using a model (301) of the application and at least one controller (302-a) for manipulating the model (301); the method comprising the following steps:
providing a context (304-a) that relates to the controller (302-a), wherein the context (304-a) has a design time context structure for storing design time declarations of the application data and further has a runtime context data structure generated from the design time context structure;
accessing a structure element (B) of the runtime context data structure, wherein the structure element (B) has a node collection (NC(B));
evaluating the node collection (NC(B));
querying a computer system (901) if the result of the evaluating step requires to fill at least one element of the node collection (NC(B)); and
in response to the querying step receiving from the computer system (901) at least one data instance that is used to fill the at least one element of the node collection (NC(B)).

18. The method of claim 17, wherein in the accessing step the structure element (B) is accessed through a typed context application programming interface.

19. The method of claims 17 or 18, wherein the computer system (901) is a system selected from the group of database system and backend system.

20. The method of any one of the claims 17 to 19, wherein the result of the evaluating step indicates that at least one element of the node collection (NC(B)) is obsolete.

21. The method of any one of the claims 17 to 20, comprising the further steps:
designating a lead selection element of the node collection (NC(B)), wherein the lead selection element has a further node collection (NC(C)) that refers to a further structure element (C) of the runtime context data structure; the further structure element (C) being a child of the structure element (B); and
updating the further node collection (NC(C)) dependent on the lead selection element.

22. A computer program product comprising instructions that when loaded into a memory of a computer system (900) cause at least one processor of the computer system (900) to perform any of the steps of the claims 13 to 16.

23. A computer program product comprising instructions that when loaded into a memory of a computer system (900) cause at least one processor of the computer system (900) to perform any of the steps of the claims 17 to 21.

24. A computer program product comprising instructions that when loaded into a memory of a computer system (900) cause at least one processor of the computer system (900) to perform according to the claims 1 to 12.

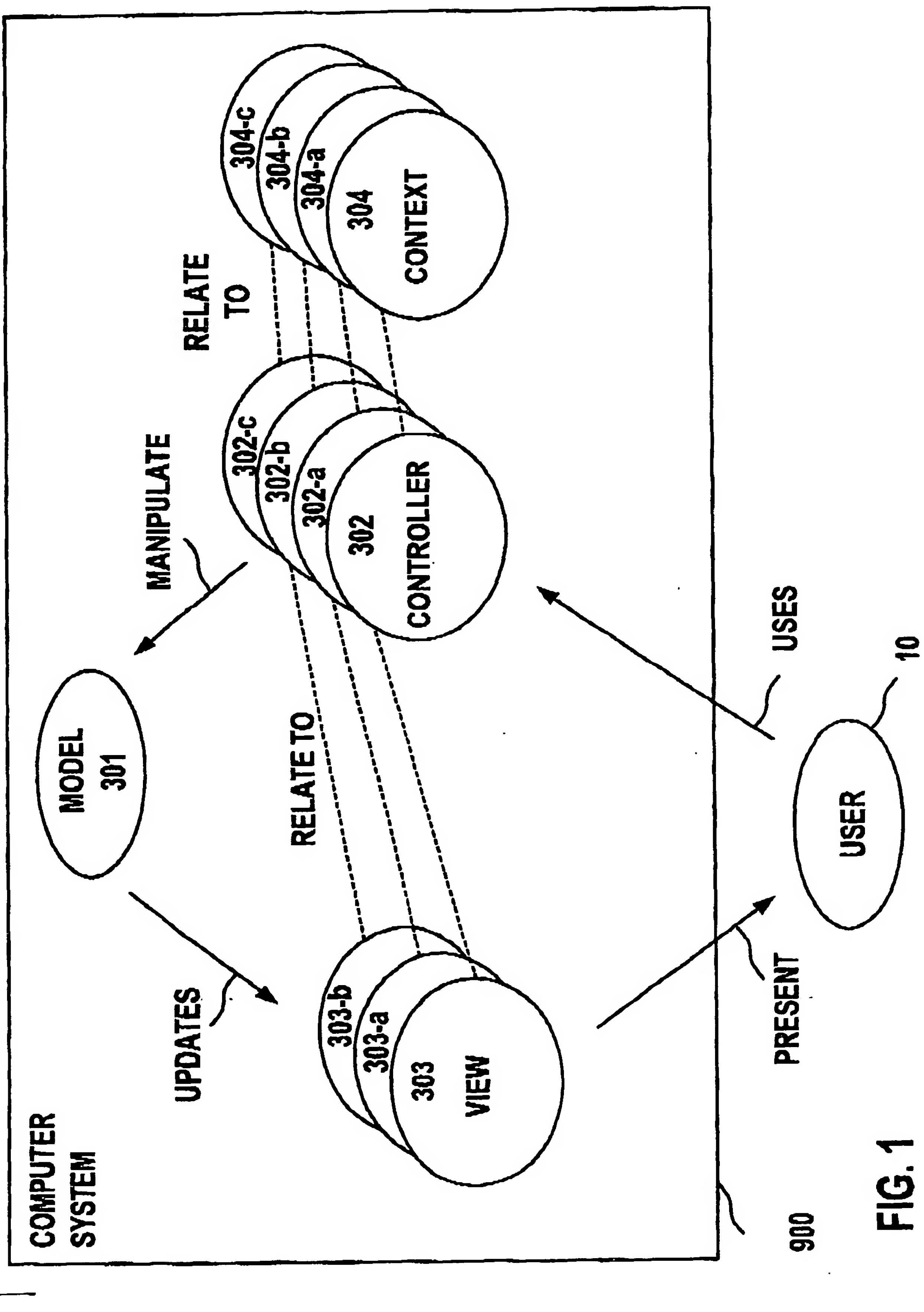

COMPUTER SYSTEM
MODEL 301
302 CONTROLLER
302-a
302-b
302-c
304 CONTEXT
304-a
304-b
304-c
303 VIEW
303-a
303-b
MANIPULATE
RELATE TO
RELATE TO
UPDATES
USES
PRESENT
USER
10
900

FIG. 1

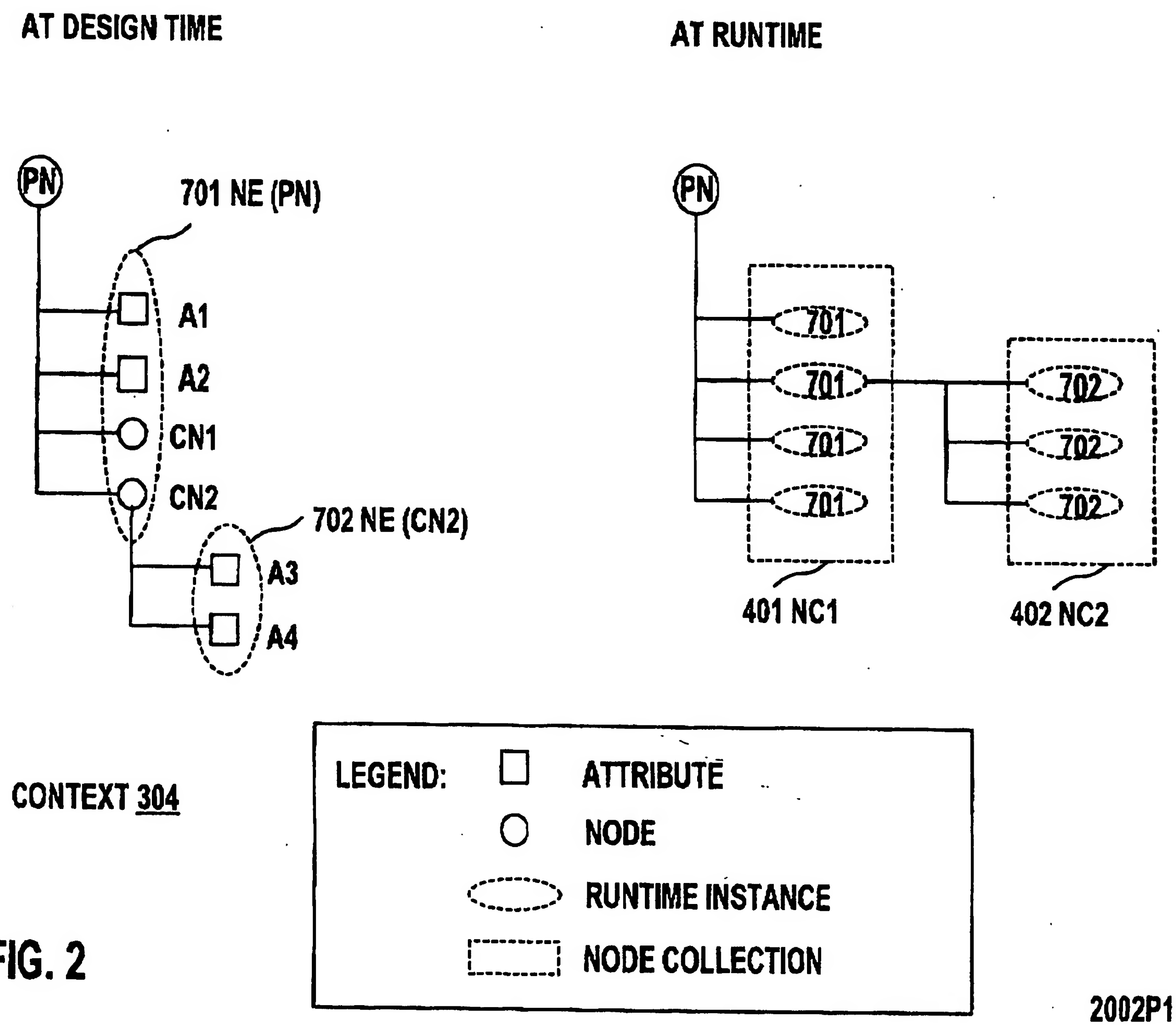


FIG. 2

2002P10032EP01

2002P10032EP01

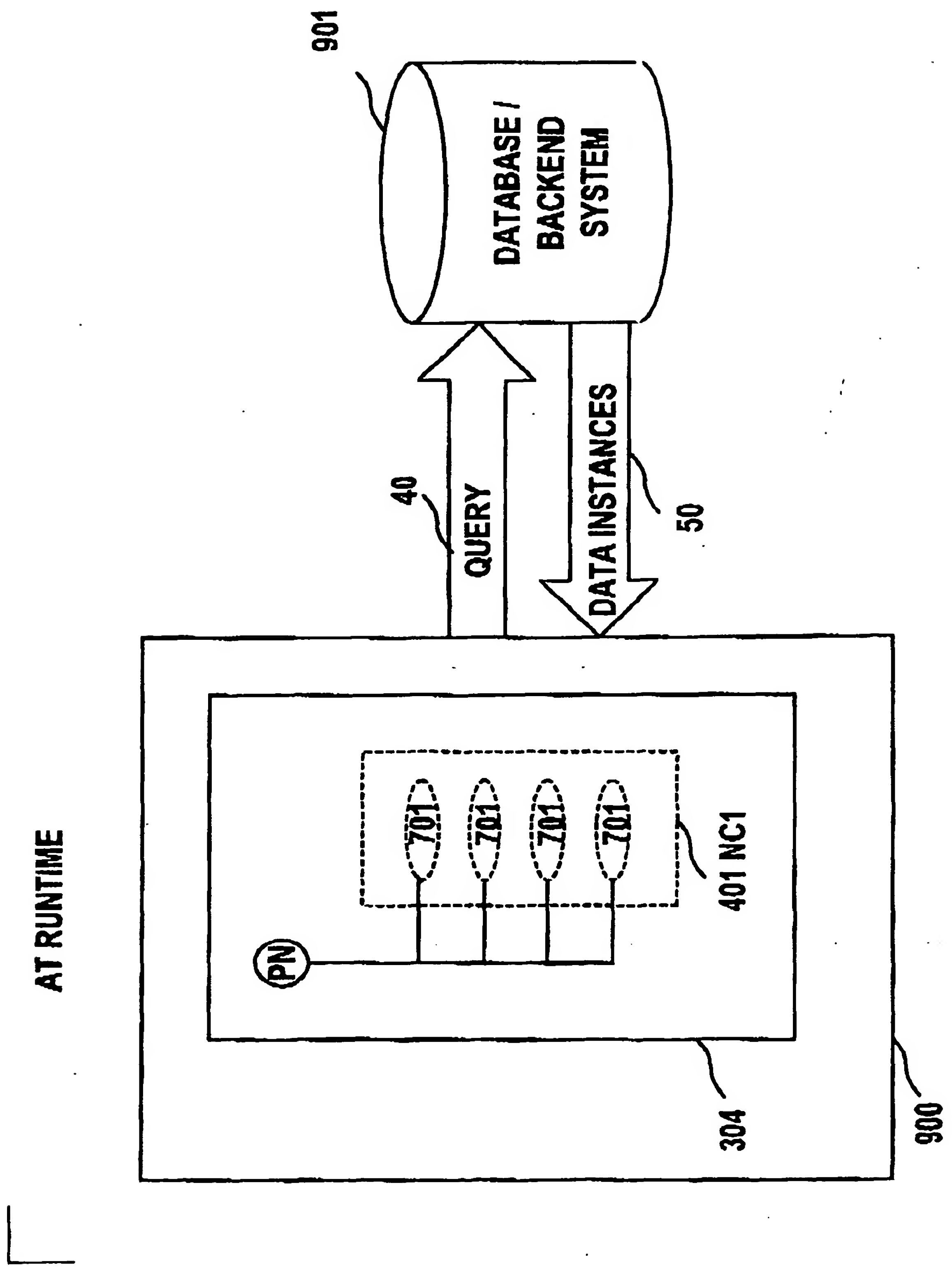


FIG. 3

2002P10032EP01

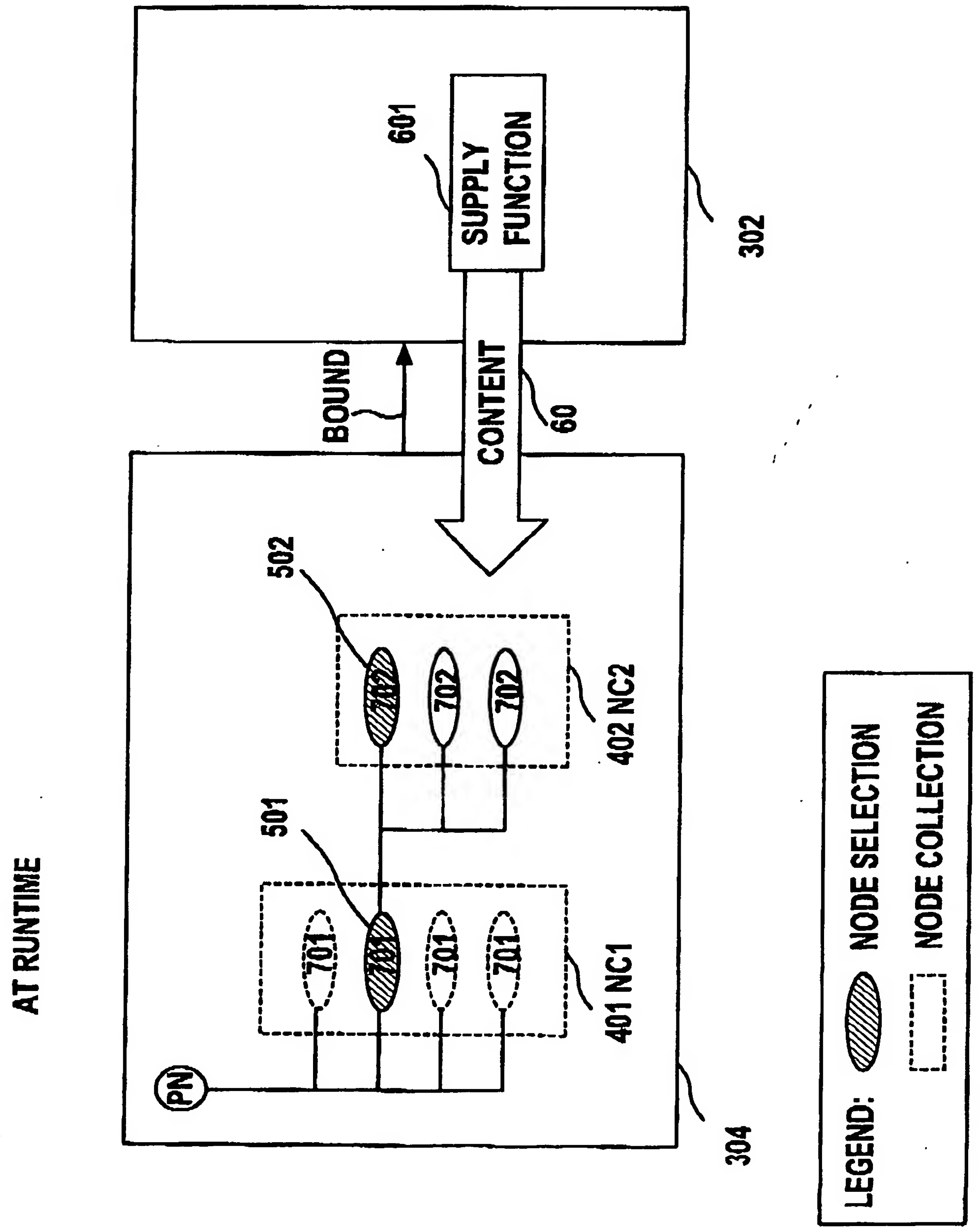
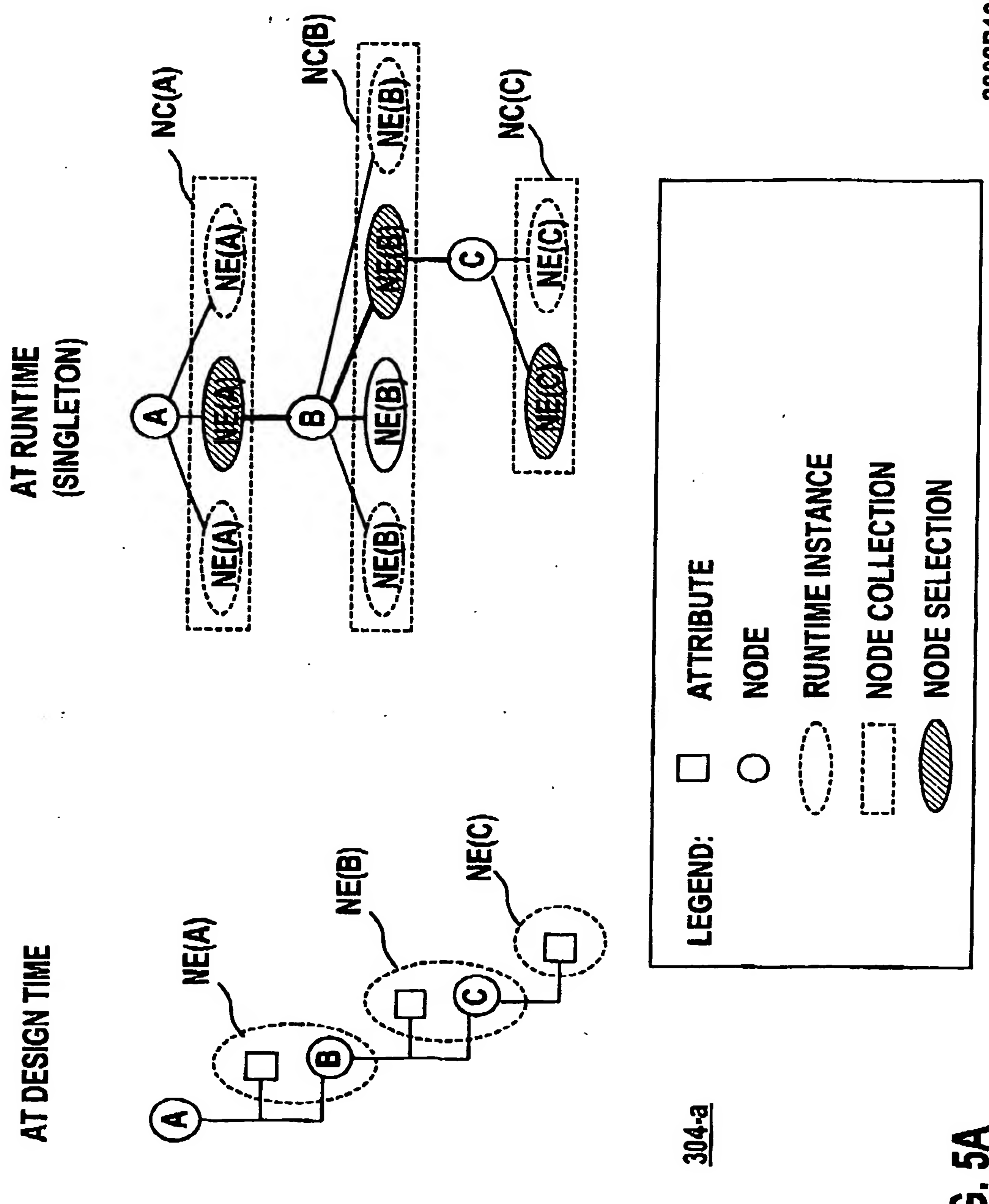


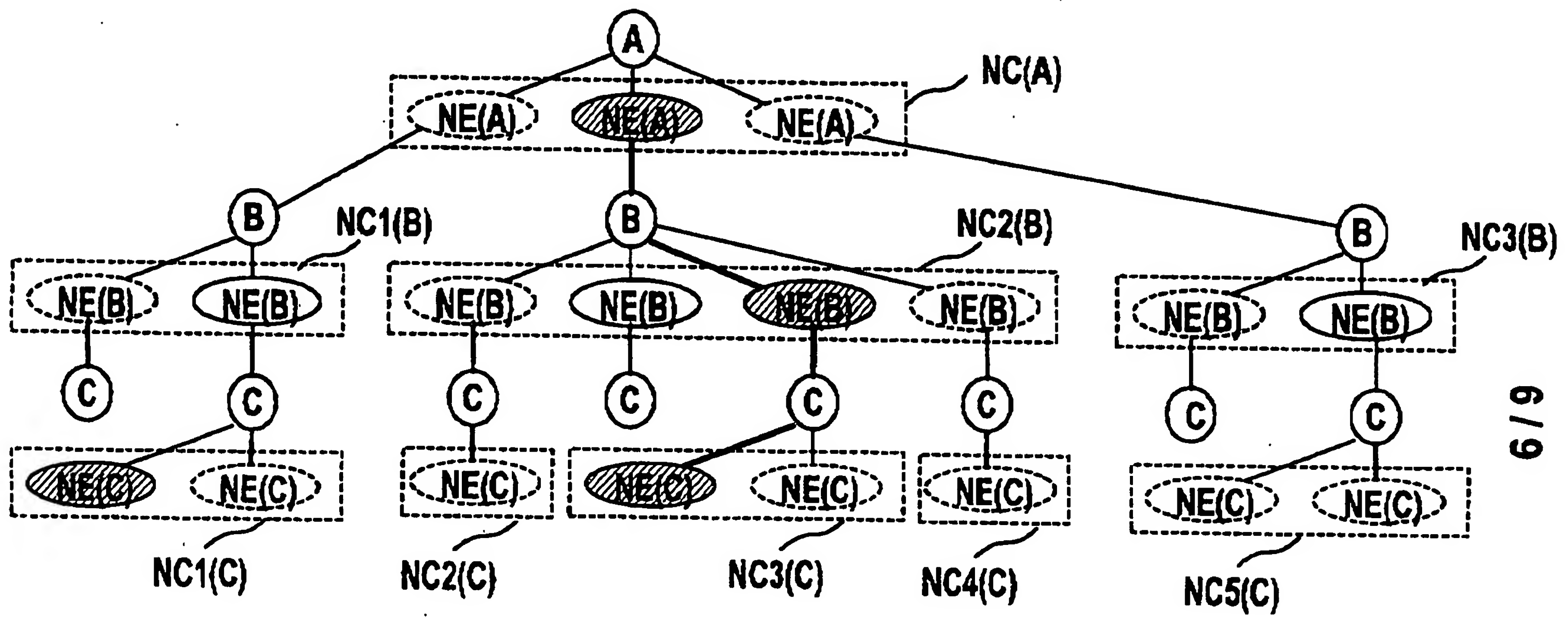
FIG. 4


AT DESIGN TIME
A
B
C
NE(A)
NE(B)
NE(C)
AT RUNTIME (SINGLETON)
NC(A)
NC(B)
NC(C)
NE(A)
NE(B)
NE(C)
LEGEND:
ATTRIBUTE
NODE
RUNTIME INSTANCE
NODE COLLECTION
NODE SELECTION

304-a

FIG. 5A

AT RUNTIME (NON-SINGLETON)



304-a

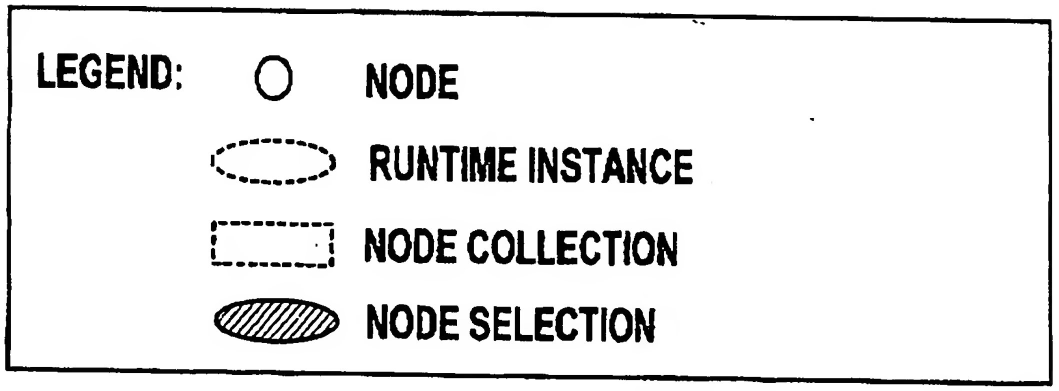


FIG. 5B

2002P10032EP01

2002P10032EP01

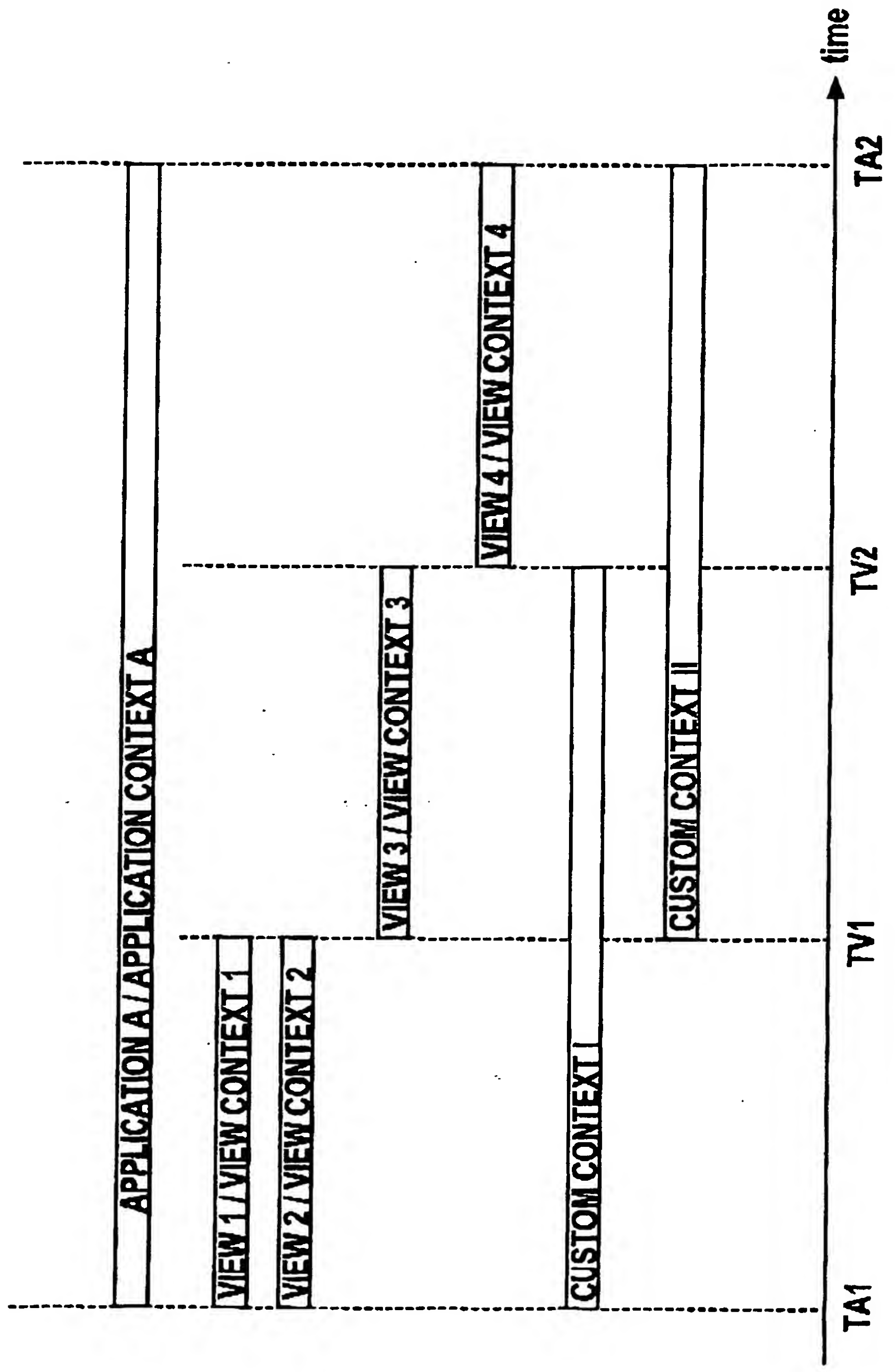


FIG. 6

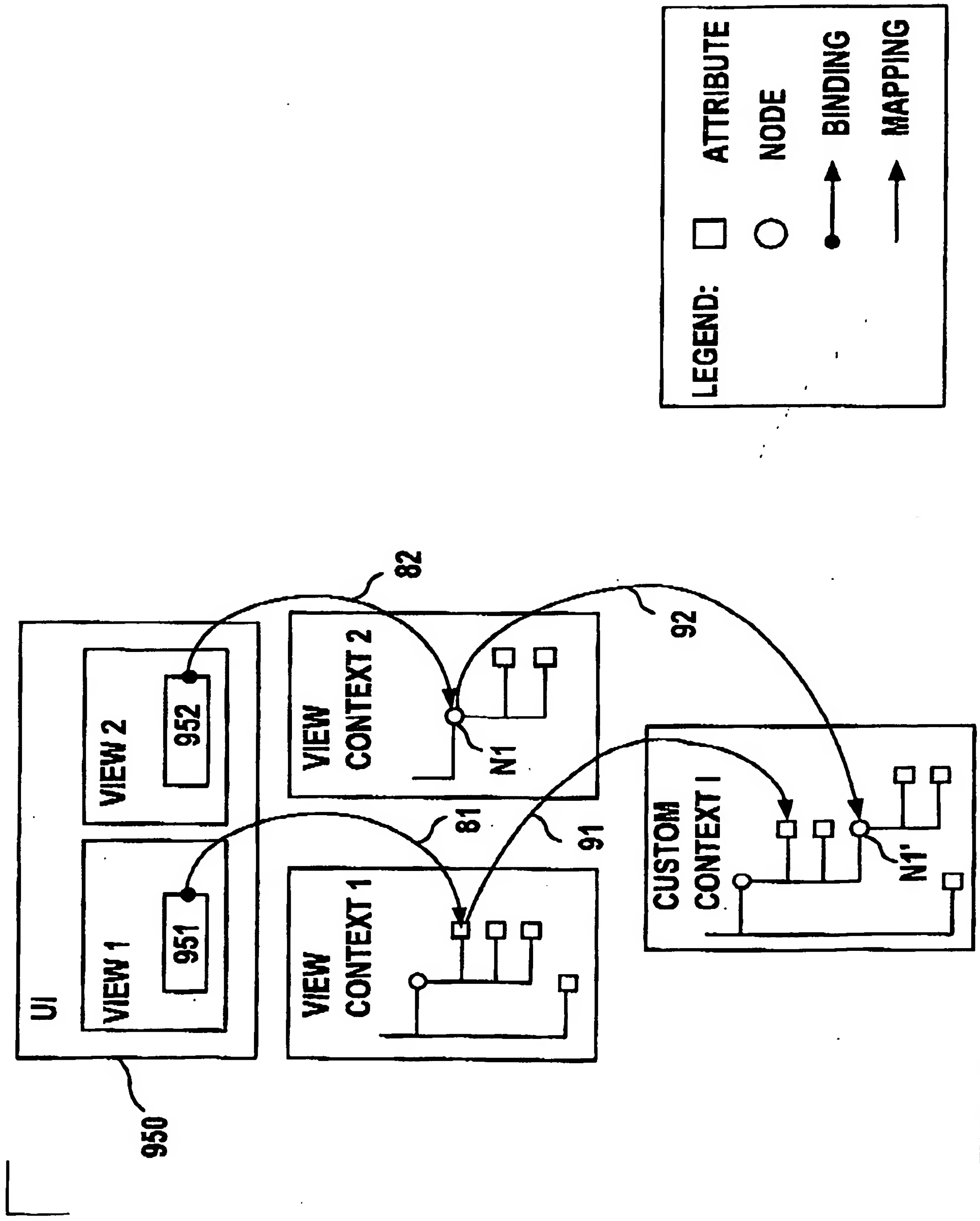

LEGEND:
ATTRIBUTE
NODE
BINDING
MAPPING
UI
VIEW 1
951
VIEW 2
952
VIEW CONTEXT 1
VIEW CONTEXT 2
N1
CUSTOM CONTEXT I
N1'
81
82
91
92
950

FIG. 7

2002P10032EP01

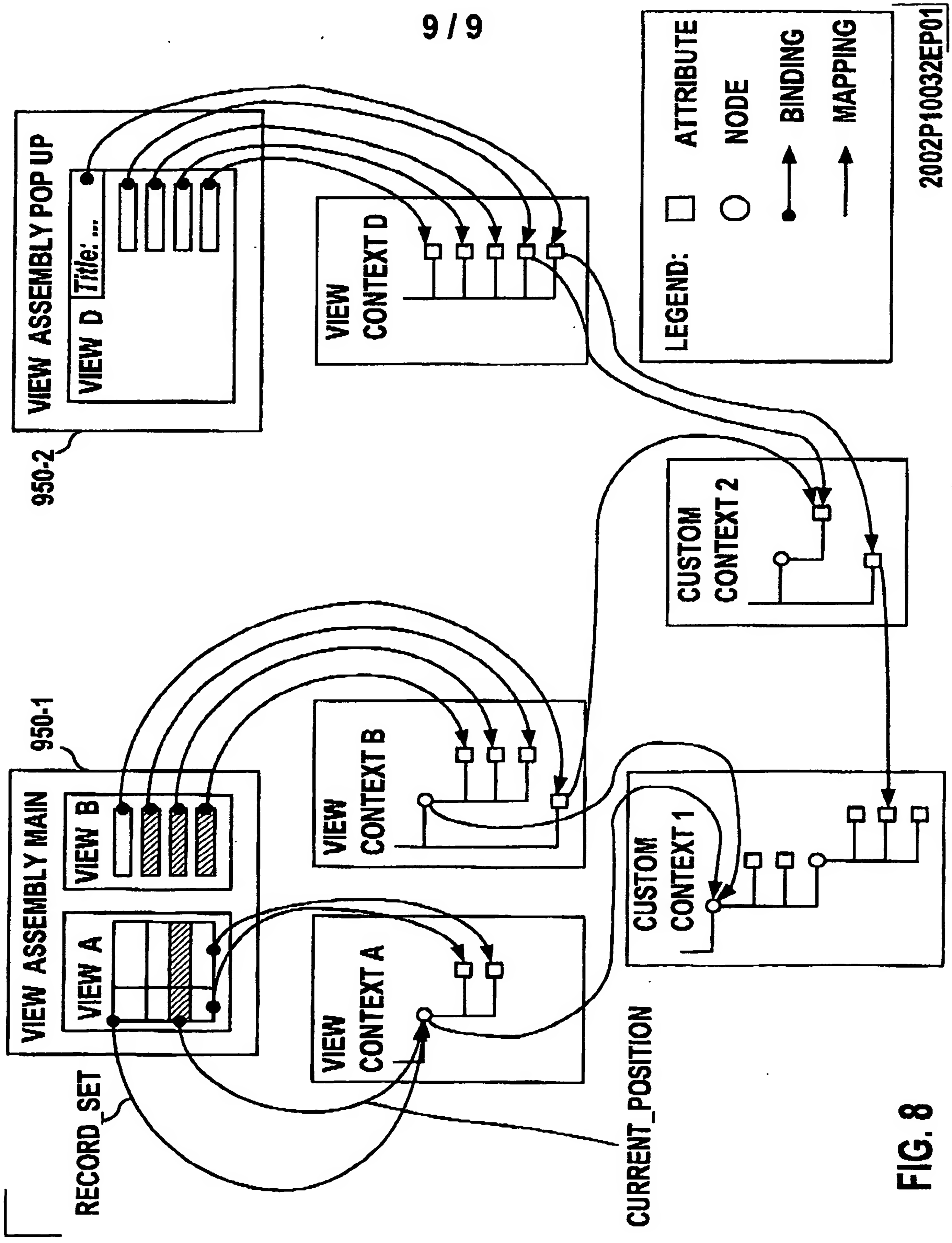


FIG. 8

COMPUTER SYSTEM AND METHODS FOR USING INTERRELATED DATA AT RUNTIME

Abstract of the Invention

A computer system (900) comprises a model (301), at least one view (303) and at least one controller (302). The at least one view (303) presents the model (301). The at least one controller (302) relates to the at least one view (303) and manipulates the model (301). The computer system (900) further comprises at least one context (304) that relates to the at least one controller (302). The at least one context (304) provides data of an application to the view (303), at runtime and comprises at least a first node collection (401) that comprises at least one runtime instance of a node (PN) of the context (304).

FIG. 1